An Effective Forwarding Architecture for SMP Linux Routers

Raffaele Bolla, Roberto Bruschi

Department of Communications, Computer and Systems Science (DIST), University of Genoa,
Via Opera Pia 13, 16145 Genova, Italy
{raffaele.bolla, roberto.bruschi}@unige.it

Abstract— **Recent technological advances provide an excellent opportunity to achieve truly effective results in the field of open Internet devices based on PC COTS Hardware, also known as Open Routers. In this environment, new interesting research topics, like parallel computing optimization, are related to the rapid migration of CPU architectures to the multi-core paradigm. Thus, with this idea, this contribution focuses on the packet forwarding performance in SMP Linux kernels, trying to identify and to characterize its architectural bottlenecks. Starting from this analysis, we propose an innovative architecture to optimize the forwarding performance when multi-processor or multi-core systems are used.**

I. INTRODUCTION

In the last years, network equipment architectures based on Open Source software have received a large interest from the scientific and industrial communities. Some well-known projects (e.g., [1], [2] and [3] among the others) have been undertaken over the last few years to develop a complete IPv4 routing platform, namely Open Router (OR), based on Open Source Software (SW) and COTS Hardware (HW). The attractiveness of the OR solution can be summarized as: multi-vendor availability, low-cost and continuous updating of the basic parts, and the flexibility of SW solutions.

Some interesting works regarding ORs can be found in the scientific literature. Among the others, Bianco et al. report in [4], [5] and [6] a detailed performance evaluation. References [7] and [8] propose a router architecture based on PC cluster, while [9] reports some performance results (in packet transmission and reception) obtained with a PC Linux testbed. In our previous works, like [10], [11], [12] and [13], we have focused our attention on testing, optimizing and evaluating the performance of the basic packet forwarding functionalities. To this purpose, besides classical external (throughput and latency) measurements, we have adopted profiling tools to optimize and to analyze in depth the internal behavior of a Linux based OR.

However, many previous works concerning Linux OR show that large interesting areas still require a deeper investigation. In particular, as shown in [13], the forwarding performance levels provided by Linux SMP kernels are usually poor, and generally lower with respect to the uni-processor (UP) versions. In greater detail, the results reported in [13] underline how much limited is the standard Linux kernel ability of effectively parallelizing the packet forwarding process, and then of exploiting the overall computational capacity of PC multi-processor systems. Nowadays and in the next future, this can be certainly considered as a heavy and critical limitation to performance scaling. In fact, it is well-known that COTS processor architectures are rapidly adopting the multi-core paradigm, instead of increasing their internal clock or their computational capacity, as suggested by the Moore's law. This trend arise out of the technical limitations in building processors with circuit dimensions much smaller than 90 - 65 nanometers. Thus, since focusing on squeezing more speed out of a single processor could be a dead end, chip markers are looking at a new class of processor [14], in which from 2 to 16 cores divide and conquer the load. In this way, no one core has to operate at hyperspeed. Each core can run much slower, but by working together, the total computational capacity of the processor is increased. However, to exploit this increased computational capacity, we need a suitable SW platform that allows an effective packet parallel processing.

In this respect, the main aim of this contribution[1] is to study and to analyze in depth why forwarding process in SMP Linux kernels provides low performance, and to introduce an innovative architecture to avoid this performance decay.

The paper is organized as follows. Section II reports a description of networking architecture of standard Linux kernels, while the proposed architecture can be found in Section III. Sections IV and V describe the used benchmarking tools and the obtained performance results. Conclusions are in Section VI.

II. THE PACKET FORWARDING ARCHITECTURE

In a PC-based router, the packet forwarding process can be substantially divided into two parts: the packets' transfer between network boards and the RAM, and the L2/L3 packet processing operations. The first phase of forwarding process is directly driven by PC hardware, since network boards (placed on I/O busses like PCI, PCI-X and PCI Express) generally use the Direct Access Memory (DMA) mechanism. The DMA mechanism allows devices to transfer data without subjecting the CPU to a heavy workload, and without any OS operations. More in details, the DMA mechanism works both in upstream and in downstream, moving data among board buffers and specific RAM areas. When an upstream or downstream data transfer is completed by DMA, the board generally signals it

[1] This work was supported by the PRIN Building Open Router Architecture Based On Router Aggregation (BORA-BORA) project financed by the Italian Ministry of Education, University and Research (MIUR).

to the CPU(s) with an IRQ. The second phase (i.e., packets' processing) is carried out by the OS, which, once advised by the IRQ, looks at the DMA areas in RAM, and starts processing the arrived packets and de-allocating the transmitted ones. In particular, the packet processing operations strictly depend on the used OS, but they are usually performed by the following steps: accessing to received data in the RAM, retrieving all the information needed in forwarding operations (e.g., L2/3 packet headers), performing the real packet forwarding operations, and delivering packets toward the selected output port. When a packet is finally delivered toward the output port, the OS inserts the RAM region where the packet is placed in the DMA memory mapping of the output device [15]. Therefore, all the packets included in this memory region are transferred to the network board with the DMA, and the transmitted. This architecture is known to provide zero-copy implementations of peripheral device drivers as well as high performance packet forwarding.

Let us now go into the details of how this networking operations are realized inside the Linux architecture in sub-Section II.A, and how the SMP architecture impacts on it in sub-Section II.B.

A. The Linux Data Plane Architecture

The Linux networking architecture is basically based on an interrupt mechanism: network boards signal the kernel upon packet reception or transmission through HW interrupts. Each HW interrupt (IRQ) is served as soon as possible by a handling routine, which suspends the operations currently being processed by the CPU. The IRQ handlers are designed to be very short, while all the time-consuming tasks are performed by the so-called "Software IRQs" (SoftIRQs) afterwards [16]. SoftIRQs differ from HW IRQs mainly in that a SoftIRQ is scheduled for execution by a kernel activity, such as an HW IRQ routine, and has to wait until it is called by the scheduler. SoftIRQs can be interrupted only by HW IRQ routines. In detail, the forwarding process is triggered by an HW IRQ generated from a network device, which signals the reception or the transmission of packets. Then the corresponding routine performs some fast checks, and schedules the correct SoftIRQ, which is activated by the kernel scheduler as soon as possible. When the SoftIRQ is finally executed, it performs all the packet forwarding operations. Thus, the forwarding process is basically performed during SoftIRQs, and can be organized in a chain of three different modules: a "reception API" that handles packet reception (NAPI [17]), a module that carries out the IP layer elaboration and, finally, a "transmission API" that manages the forwarding operations to the egress network interfaces. In particular, the reception and the transmission APIs are the lowest level modules, and are activated by both HW IRQ routines and scheduled SoftIRQs. They handle the network interfaces and perform some layer 2 functionalities. Major details of Linux architecture can be found in [10].

B. The Linux SMP architecture

Multiprocessing architectures, like SMP and NUMA, allow to considerably increase the computational capacity of PCs, since they allow multi-CPUs/cores usage, where tasks can be computed in a parallel way. However, this kind of architectures generally introduces complexity in both the hardware and the software levels. Typical issues in parallel processing include the serialization of tasks (i.e., code locking [18]) and the coherence of shared data in cache [18]. In the particular, these two aspects may lead to a not effective utilization of the overall computational capacity; notwithstanding this, a wise SW multiprocessing support can certainly reduce their effects in many cases.

With regard to Linux and SMP support, the 2.6 kernels have reached a high optimization level. The key of SMP improvements was the ability to load balance work across the available CPUs while maintaining some affinity (i.e., the capacity of binding a process to a specific CPU) for cache efficiency[2] [19]. More in particular, the CPU affinity is very important to avoid the ping-pong effect, and to consequently maximize CPU cache efficiency: when a task moves from a CPU to another, its cache flushes with it. This increases the latency of the task's memory access until its data is in the cache of the new CPU, while cache miss rates grow very large. CPU affinity protects against this performance decay and improves cache hit rate.

Similar remarks can be made also on data shared among more processes. If processes run on the same CPU, they use the same cache, and access to up-to-date shared data with no problems. On the contrary, if processes run on different CPUs, data can be kept in only one processor's cache at a time, otherwise the processor's cache may grow out of synchronization (i.e. some processors may work on not up-to-date data). Consequently, whenever a processor adds a line of data to its local cache, all the other processors also caching it must invalidate this line. This invalidation is costly.

Moreover, Linux allows to assign certain IRQs to specific processors. This is known as "SMP IRQ affinity", and allows to control how the system responds to various HW events, and to balance or to repartition IRQ load among processors. The IRQ affinity has a strong impact on forwarding process in SMP kernels. In fact, when a board IRQ is bound to one or more CPUs, the corresponding HW IRQ handler and the following SoftIRQ(s) will be always kept on those CPUs. As a board rises its IRQ, during the corresponding handler the kernel starts working on two main tasks:

- *TxRing management operations*: they include the access to the TxRing and de-allocation of packets successfully transmitted by the board generating the IRQ;
- *Packet forwarding operations*: these operations start fulfilling standard NAPI procedures (for the packets received by the board generating the IRQ), and include also all the following delivery operations. All the packets kept with NAPI will be processed (in the same SoftIRQ and by the same CPU) up to the enqueuement in the TxRing of output network interfaces.

[2] L2 cache, local to each processor.

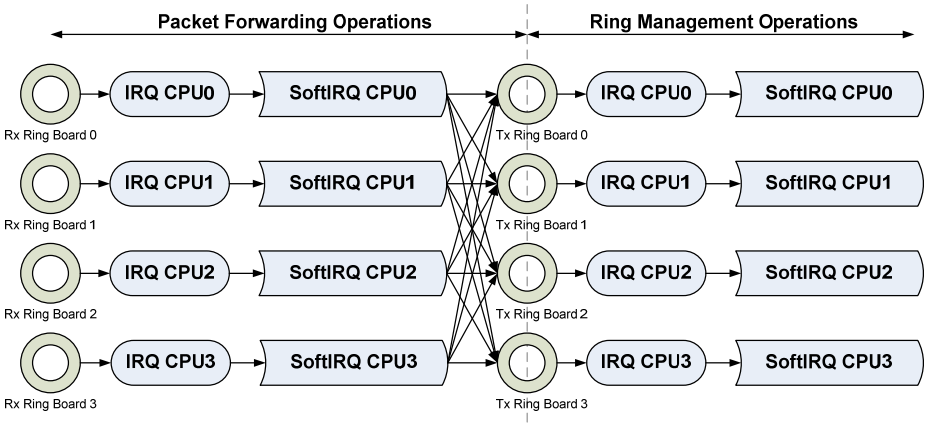

Fig. 1. Example of CPU load balancing in a standard SMP kernel: 4 network boards are bound to 4 different CPUs.

Thus, while the RxRing data structure of a specific network interface is accessed and modified only by bound CPUs, TxRing can be modified by every CPUs. More in particular, the TxRing of a board can be filled with packets by any CPUs delivering traffic toward it, while it can be emptied only by the bound CPUs, since de-allocation operations are performed as consequence of the Tx board IRQ. Therefore, TxRing of network interfaces can be considered as the critical point of the SMP forwarding process, since it is a data structures shared among all CPUs involved in packet delivery.

With the aim of deeper analysing these concepts, let us to consider the case in Fig. 1, where 4 network interfaces are assigned to 4 different CPUs. With such setup we expect to uniformly balance the OR work among CPUs. In this environment, the packets received from the generic interface *i* are processed and delivered toward any other interfaces by CPU *i*. Thus, while CPU *i* can backlog packets into the TxRings of all the network boards, these last ones are periodically cleaned by their bound CPUs, which generally differ from CPU *i*. As shown in Section V, this causes a large number of cache misses in all the CPUs, and consequently a so heavy performance waste, as to cancel all the positive effects due to the presence of more parallel CPUs .

III. THE PROPOSED ARCHITECTURE

To make optimum use of SMP architectures, the considerations included in the sub-Section II.B clearly suggest that we must exploit the CPU affinity in the forwarding process as much as possible. With this aim, the main objectives of the proposed architecture are substantially two: to entirely bind to a single CPU all the operations carried out to forward a packet, and to equally distribute the computational load among all the processors/cores in the system. As shown in sub-Section II.B, standard Linux kernels cannot achieve these two objectives at the same time: when input and output interfaces are bound to different CPUs, we have a high load distribution, but a large number of cache misses/invalidations too. When all the interfaces are bound to a CPU, we obviously have no cache invalidations, but we exploit only a single core.

The base idea of the proposed architecture is basically to maintain separated as much as possible those memory regions, which are accessed by the forwarding processes running as SoftIRQ on different CPUs. This is not a simple task, since the TxRing of standard network interfaces can be accessed by every CPU as shown in the example of Fig. 1. Moreover, in standard Linux/COTS hardware environments, it is not possible to provide TxRings where each CPU has exclusive access to received or transmitted packets. Thus, our idea is simply to provide a certain number of TxRings per device; ideally a number of TxRings equal to the number of CPUs involved in the forwarding process. Each TxRing of the same device has to be assigned to a different CPU: i.e. only a single CPU can access and modify this memory region. In this way, when a packet is received and processed by a CPU, it will be placed in the corresponding TxRing. When the packets on a TxRing are successfully transmitted, just the CPU bound to that Ring can clean them. More in the detail, to notice the successfully packet transmissions to the right CPUs, each device needs to raise a number of HW IRQ channels equal to the number of the included TxRings, since each of them is bound to a specific CPU. Each HW IRQ assigned to a device allows to trigger the forwarding operations on a different CPU.

This solution allows to realize separated TxRings and to avoid cache invalidates as much as possible. With the aim of better explaining this architecture, let us consider the case of a SMP OR with two CPUs and two network interfaces. In such case, as shown in Fig. 2, each network interface has one RxRing and two TxRings. Let assume that the RxRing of board 0 is assigned to CPU 0 (i.e., when the board 0 receives a packet, it will raise a HW IRQ that can be kept only by the CPU 0), while the one of board 1 is assigned to CPU 1. The TxRing 0 of each board is assigned to CPU 0, while the TxRing 1 to CPU 1. Then, the packets received by the board 0 are processed by SoftIRQs running on the CPU 0 with standard NAPI operations, and they can be theoretically forwarded toward whichever network interface in the system.

Once the output interface has been selected, packets need to be enqueued into the right TxRing accordingly to their processing CPU: in particular, with reference to the example in Fig. 2, packets processed by CPU 0 will be backlogged to TxRing 0 of board 0 or 1.

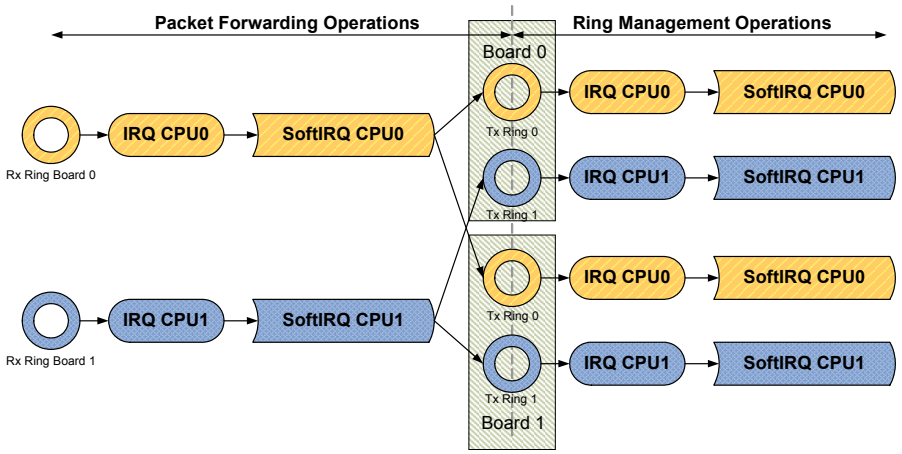

Fig. 2. The proposed architecture in the case of 2 CPUs and 2 network boards. Both the network cards include two TxRings, one per CPU. Note that in the proposed architecture all the TxRing are accessed and modified just by a single CPU.

When a board transmits one or more packets of TxRing 0, it raises a HW IRQ bound to CPU 0, then the descriptors of transmitted packets are de-allocated by CPU 0 during the following HW IRQ and SoftIRQ handlers. Similar considerations can also be made for the TxRings assigned to CPU 1. As shown in Section 0, this kind of architecture allows to considerably boost the forwarding performance, but it requires some new modifications/developments both in SW and in HW: e.g., to provide a certain number of TxRing per device, we need particular network cards with advanced DMA engine, able to manage more DMA regions and to raise different IRQ channels. However, this network card architecture does not considerably differ from the one commonly used to realize multi-port devices like [20] and [21]: the only main difference is the number of interfaces and MAC Hardware elements, since these network cards generally support DMA engines with the all advanced features that our architecture needs. For what concerns the software, the proposed architecture requires only few minor changes in the "advanced" network device drivers with more TxRings. In particular, we have only to assure that each CPU accesses to its own TxRing in backlogging and in de-allocating methods. Moreover, a careful Software architecture can easily maintain the compatibility with standard network devices supporting only one TxRing.

IV. BENCHMARKING SCENARIO

Since we do not have any prototypes yet of the advanced network board described in Section V, we decided to emulate it by using quad-port network boards and some Gigabit Ethernet (GE) switches. More expressly, we used each quad-port board to "emulate" a simple GE interface with the separated TxRings and the HW IRQ channels per CPU, and we used the L2 switches to aggregate traffic delivered by this emulated output interface. For example, in the testbed reported in sub-Section V.B with 3 CPUs and 3 network adapters, in spite of using 3 "advanced" network boards, we utilized 3 quad-port Gigabit adapters, each one with 3 interfaces to emulate the needed TxRings and HW IRQ channels; therefore, the traffic coming from the 3 interfaces on the same quad-port adapter has been aggregated with an Ethernet switch.

The Gigabit quad-port adapters belong to the Intel Gigabit Adapters' family [20], and are equipped with PCI-e controllers. For what concerns the other OR Hardware elements, we used two Intel based architectures, which can be regarded as a high-end server and a workstation, respectively. In the particular, the server architecture includes two 64bit 3.0GHz Dual Core Xeons, equipped with 2MB of L2 cache per core, and a X7DB8 SuperMicro mainboard with both PCI-X and PCI-e bus slots. The workstation architecture is based on a 3.0GHz Dual Core Pentium-D processor with 2MB of L2 cache per core, and on a P5B-VM Asus mainboard with two PCI-e slots.

To benchmark the OR forwarding performance, we used a professional device, known as Agilent N2X Router Tester [22], which can be used to obtain throughput and latency measurements with high availability and accuracy levels (i.e., the minimum guaranteed timestamp resolution is 10 ns).

We also performed some internal measurements using specific software tools (called profilers) placed inside the OR. In the particular, we utilized Oprofile [23], an open source tool that continuously monitors system dynamics with frequent and quite regular sampling of CPU hardware registers. Oprofile effectively and profoundly evaluates CPU performance counters (e.g., like L2 cache misses and hits) and the utilization of each software application and each single kernel function running in the system with very low computational overhead.

V. NUMERICAL RESULTS

This Section reports a set of the performance results obtained by studying and analyzing the OR behavior in the presence of the standard and the proposed SMP architectures in different benchmarking environments. In greater detail, both the used Software architectures are based on the optimized 2.6 Linux kernel, which includes the "skb recycling patch" and other ad-hoc optimizations, already studied and introduced in our previous works [10], [11], [12] and [13].

This Section is organized in two different parts. The first one describes the forwarding performance when a standard SMP Linux is used, and shows different internal measurements with the aim of characterizing the SMP bottlenecks. In the second sub-Section, we report the performance results obtained with the proposed new architecture, and we compare them with the the standard SMP Linux ones.

A. Forwarding Performance of the Standard SMP Kernel

This benchmarking session consists in a forwarding performance evaluation of standard SMP Linux kernel. Our main aim is to analyze its behavior and to characterize its internal performance bottlenecks. At this purpose, we study the behavior of two different SMP kernel setups in a very simple environment: we consider only a traffic flow crossing the OR among two Gigabit Ethernet interfaces. The packet size is fixed to 64B, since the computational capacity (i.e., the maximum number of packet headers that can be computed per second) is the major performance bottleneck [10].

The used kernel setups are: SMP 1-CPU: in this kernel configuration, both the crossed network adapters are bound to the same CPU; SMP 2-CPU: in such case, the two network adapters are assigned to different CPUs. While the former (SMP 1-CPU) tries to maximize the CPU affinity of the forwarding process, the latter (SMP 2-CPU) performs a load balancing among CPUs. The performance results obtained with a uni-processor (UP) kernel are used as term of comparison. Fig. 5 and Fig. 6 show the throughput and latency values obtained with all the studied kernel setups and the Xeon hardware. While the UP kernel forwards about the full gigabit (i.e., 1460 kPkt/s with 64B packets), the SMP kernel versions show lower performance: the SMP 1-CPU achieves a maximum forwarding rate equal to about 1020 kPkt/s, while the SMP 2-CPU to about 650 kPkts/s.

Fig. 3 and Fig. 4 report results similar to the ones in Fig. 5 and Fig. 6, but obtained with the Pentium-D hardware. In such case, the maximum throughput values are lower with respect to the previous case, but their trends look very similar to the previous ones: the UP kernel provides the best forwarding performance, while the SMP 2-CPU setup the lowest one. We can justify the performance gap among SMP and UP kernels with the additional complexity overhead required in SMP kernels. In fact, as shown in the Section II.B, multi-processor kernels, unlike the UP ones, need some advanced features to manage the concurrency in parallel processing. Therefore, given this additional overhead, the computational effort needed by SMP kernels to forward a packet is higher than the effort needed by the UP ones, and the SMP maximum throughput is consequently lower. However, there is a clear performance gap between the SMP setups, too: the performed tests show that the load balancing of the forwarding processes among different CPUs generally leads to a heavy performance decay. In short, the results obtained with these tests suggest to assign all the forwarding process to only one CPU, like in the SMP 1-CPU and the UP cases; however, these last solutions do not allow to exploit the overall computational capacity of SMP systems. Performance decays in SMP systems can be fundamentally caused by two main issues: the code serialization and the cache coherence. While cache coherence is deeply discussed in Section II.B, the code serialization is the method by which the OS guarantees the right sequential access order to a shared data. The code serialization is implemented in all Linux kernel through the "spinlock" paradigm. However, in a SMP environment, spinlocks can have a different impact with respect to the UP case. Spinlocks are "busy-wait" locks: when two kernel path executions try to overlap on the same piece of code or on a same shared data, the first-come acquires the lock, while the last one is stopped and continuously "spins on the lock", checking for its availability.

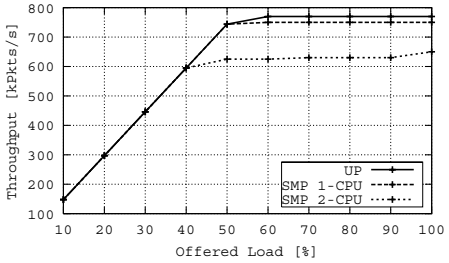

Fig. 3. Throughput values obtained with the Pentium-D based hardware architecture.

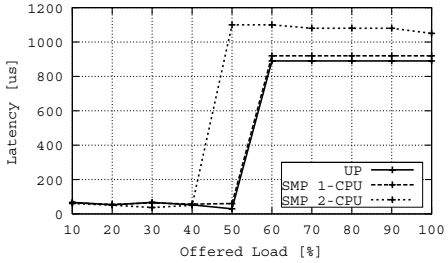

Fig. 4. Latency values obtained with the Pentium-D based hardware architecture.

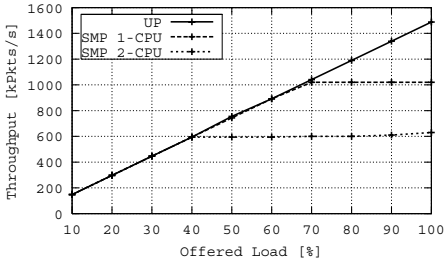

Fig. 5. Throughput values obtained with the Xeon based hardware architecture.

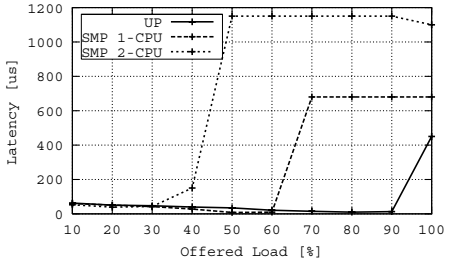

Fig. 6 Latency values obtained with the Xeon based hardware architecture.

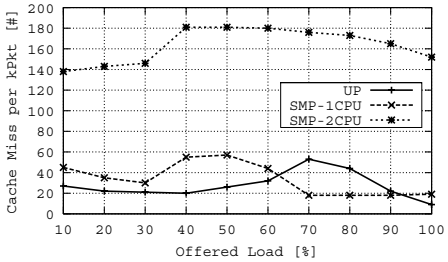

Fig. 7. Xeon cache misses per forwarded Kpackets.

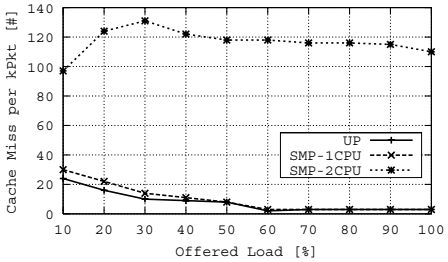

Fig. 8. Pentium-D cache misses per forwarded Kpackets.

Fig. 7 and Fig. 8 show the number of cache misses obtained with Oprofile for all the considered three SW setups and the Xeon and Pentium-D based architectures, respectively. Observing these figures, we can underline how the number of cache misses is notably larger in the SMP 2-CPU setups. This confirms what written about the example shown in Fig. 1: when packets cross the OR among interfaces bound to different CPUs, they causes a large number of cache invalidates, and a consequent slowing down of memory accesses. Fig. 9 and Fig. 10 report the number of spinlocks and waiting times, respectively, when Xeon-based hardware architecture is used. The number of spinlocks is considerably large in both the SMP setups. This is an indication of how much the code parallelization is refined: a larger spinlock number generally points out small code locking regions, instead of fewer and extended ones. Fig. 10 shows an apparently strange behavior: the SMP 1-CPU setup, which provide better performance, shows larger values of spinlock waiting times with respect to the SMP 2-CPU one.

This is simply caused from the fact that access synchronization to shared data in the SMP 2-CPU case is more critical, since the forwarding process involves two independent and parallel CPUs. In the SMP 1-CPU setup, where only a CPU is involved, access synchronization is partially performed by the kernel scheduler, which sequentially runs active kernel paths.

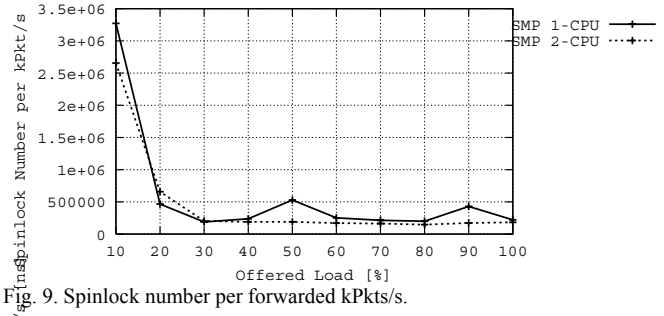

Fig. 9. Spinlock number per forwarded kPkts/s.

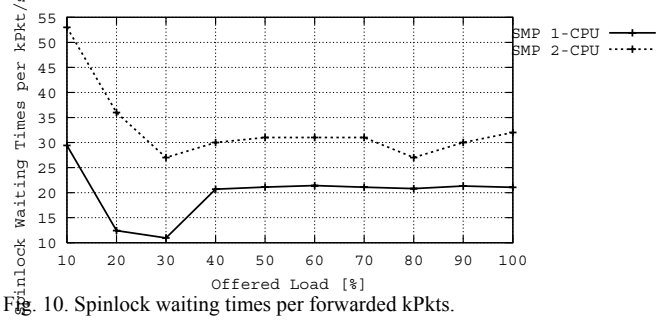

Fig. 10. Spinlock waiting times per forwarded kPkts.

B. Performance Evaluation of Proposed Kernel Architecture

This sub-Section reports the results of some benchmarking tests carried out to analyze the proposed architecture performance. We used the standard SMP Linux kernel as term of comparison. We realized four different testbeds with a increasing complexity level. In the particular, we used the Xeon based hardware architecture with a variable number of CPUs (in the following the term "CPU" is used to indicate both CPU and cores) and network boards. The number of the Gigabit interfaces has been kept equal to the number of the used CPUs, and each interface was bound to a different CPU.

For what concerns the test traffic, we used full-meshed traffic matrixes composed by CBR flows of datagrams with fixed packet size. Fig. 11 shows the maximum throughput of both the new architecture and the SMP Linux kernel, obtained with 2, 3 and 4 Gigabit Ethernet interfaces, and according to different packet sizes. From this figure, we can note how both the considered architectures achieve the full Gigabit speed only for large packet sizes. This is a clear indication of a performance bottleneck in the computational capacity, which limits the maximum OR throughput in terms of processed packet headers per second. However, in the presence of small packets, the proposed architecture clearly provides better performance in all the benchmarking scenarios. This suggests that the proposed architecture allows to reduce the computational weight per forwarded packet. In greater detail, this reduction can be associated to the different memory management that we have obtained with our proposal.

With the aim of analyzing the OR behavior thoroughly in the presence of small packet sizes, Fig. 12 and Fig. 13 report the throughput and the latency values, obtained with the datagram size fixed to 64 Bytes and the same test environment of Fig. 11. In such a case, we note how the standard SMP kernel holds about the same throughput independently from the number of active CPUs.

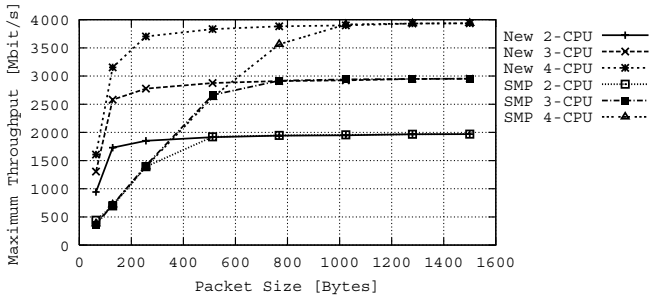

Fig. 11. Maximum throughput values for the proposed architecture and the SMP Linux kernel with according to 2, 3 and 4 Gigabit Ethernet interfaces/CPUs and variable packet sizes.

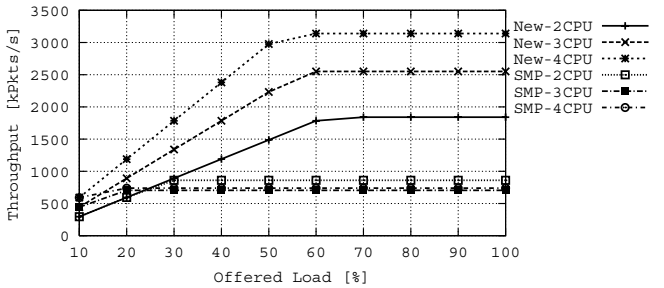

Fig. 12. Throughput values for the proposed architecture and the SMP Linux kernel with according to 2, 3 and 4 Gigabit Ethernet interfaces/CPUs, and to a variable offered load. Datagram size is fixed to 64 Bytes.

On the contrary, the proposed architecture increases its maximum performance with a linear proportion with respect to the number of active CPUs. In greater detail, Fig. 12 shows that the proposed architecture performance scales very well with the number of CPUs and Gigabit interfaces, always guaranteeing, for the smallest packet sizes, a throughput near to the 50% of the maximum link capacities. Moreover, the

maximum latency values in Fig. 13 clearly show that packet processing times considerably increase only for the standard kernels, while they are more or less constant among the all the used setups of the proposed architecture. Observing Fig. 14, we can conclude that these large packet processing times for standard kernels are referable to the high number of cache misses, which causes more and more RAM accesses as the number of CPUs increases. On the contrary, the proposed architecture allows to keep constant the number of cache misses in all the performed benchmarking tests.

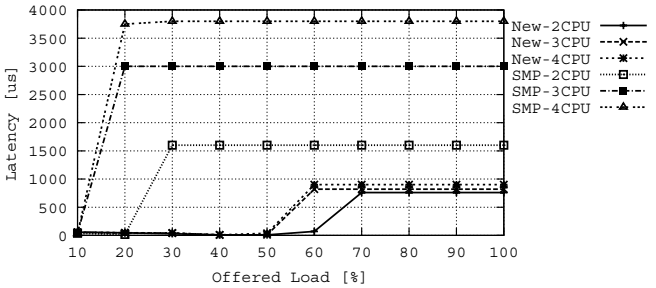

Fig. 13. Latency values obtained in the benchmarking tests of Fig. 12.

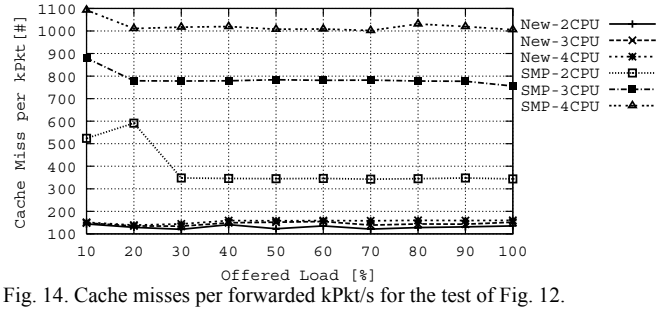

Fig. 14. Cache misses per forwarded kPkt/s for the test of Fig. 12.

VI. CONCLUSION

In this contribution we report an in-depth study of the forwarding process of a PC Open Router architecture when a SMP Linux kernel is adopted for data plane. The internal and external measurement results performed in this analysis clearly show that SMP kernels provide very low performance caused by an ineffective use of the L2 cache. Thus, we propose a new architecture to optimize the SMP performance. This new architecture, which requires minor changes both at software and hardware levels, allows to exploit a more effective "CPU affinity" in the forwarding process. The large set of reported benchmarking results shows that our proposal achieves larger throughput values with respect to the standard SMP kernels, and its performance linearly scales with the number of active CPUs. In particular, the benchmarking tests show that this new architecture achieves maximum throughput values equal to about 1800, 2550, and 3200 Kpkts/s when are used 2, 3 and 4 CPUs, respectively.

REFERENCES

[1] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek,"The Click modular router", ACM Transactions on Computer Systems 18(3), August 2000, pages 263-297.

[2] Zebra, http://www.zebra.org/.

[3] M. Handley, O. Hodson, E. Kohler, "XORP: an open platform for network research", ACM SIGCOMM Computer Communication Review, Vol. 33 Issue 1, Jan 2003, pp. 53-57.

[4] A. Bianco, R. Birke, D. Bolognesi, J. M. Finochietto, G. Galante, M. Mellia, M.L.N.P.P. Prashant, Fabio Neri, "Click vs. Linux: Two Efficient Open-Source IP Network Stacks for Software Routers", Proc. of IEEE 2005 Workshop on High Performance Switching and Routing (HPSR 2005), Hong Kong, May 12-14, 2005, pp. 18-23.

[5] A. Bianco, J. M. Finochietto, G. Galante, M. Mellia, F. Neri, "Open-Source PC-Based Software Routers: a Viable Approach to High-Performance Packet Switching", Third International Workshop on QoS in Multiservice IP Networks, Catania, Italy, Feb 2005, pp. 353-366.

[6] A. Bianco, R. Birke, G. Botto, M. Chiaberge, J. Finochietto, M. Mellia, F. Neri, M. Petracca, G. Galante, "Boosting the Performance of PC-Based Software Routers with FPGA-enhanced Network Interface Cards", Proc. of IEEE 2006 Workshop on High Performance Switching and Routing (HPSR 2006), Poznan, Poland, June 2006, pp. 121-126.

[7] B. Chen and R. Morris, "Flexible Control of Parallelism in a Multiprocessor PC Router", Proceedings of the 2001 USENIX Annual Technical Conference (USENIX '01), Boston, Massachusetts, June 2001.

[8] C. Duret, F. Rischette, J. Lattmann, V. Laspreses, P. Van Heuven, S. Van den Berghe, P. Demeester, "High Router Flexibility and Performance by Combining Dedicated Lookup Hardware (IFT), off the Shelf Switches and Linux", Proc. of Second International IFIP-TC6 Networking Conference, Pisa, Italy, May 2002, LNCS 2345, Ed E. Gregori et al, Springer-Verlag 2002, pp. 1117-1122.

[9] A. Barczyk, A. Carbone, J.P. Dufey, D. Galli, B. Jost, U. Marconi, N. Neufeld, G. Peco, V. Vagnoni, "Reliability of datagram transmission on Gigabit Ethernet at full link load", LHCb technical note, LHCB 2004-030 DAQ, March 2004.

[10] R. Bolla, R. Bruschi, "Linux Software Router: Data Plane Optimization and Performance Evaluation", Journal of Networks (JNW), Academy Publisher, vol. 2, no. 3, pp. 6-11, 2007.

[11] R. Bolla, R. Bruschi, "The IP Lookup Mechanism in a Linux Software Router: Performance Evaluation and Optimizations", Proc. of the 2007 IEEE Workshop on High Performance Switching and Routing (HPSR 2007), New York, NY, USA, May 2007.

[12] R. Bolla, R. Bruschi, "RFC 2544 Performance evaluation of a Linux Based Open Router", Proc. of the 2006 IEEE Workshop on High Performance Switching and Routing (HPSR 2006), Poznan, Poland, June 2006, pp. 9-14.

[13] R. Bolla, R. Bruschi, "IP forwarding Performance Analysis in presence of Control Plane Functionalities in a PC-based Open Router", Proc. of the 2005 Tyrrhenian International Workshop on Digital Communications (TIWDC 2005), Sorrento, Italy, Jun. 2005, and in F. Davoli, S. Palazzo, S. Zappatore, Eds., "Distributed Cooperative Laboratories: Networking, Instrumentation, and Measuments", Springer, Norwell, MA, 2006, pp. 143-158.

[14] D. Geer, "Chip makers turn to multicore processors", Computer, vol. 38, no. 5, pp.11 – 13, 2005.

[15] J. Corbet, A. Rubini, G. Kroah-Hartman, "Linux Device Drivers", 3rd ed., O'Really Media Inc., Sebastopol, CA 2005.

[16] K. Wehrle, F. Pählke, H. Ritter, D. Müller, M. Bechler, "The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel", Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2004.

[17] J. H. Salim, R. Olsson, A. Kuznetsov, "Beyond Softnet", Proc. of the 5th annual Linux Showcase & Conference, Nov. 2001, Oakland, California, USA.

[18] R. Love, "Kernel locking techniques", Linux Journal, 2002, url: http://www.linuxjournal.com/article/5833.

[19] R. Love, "CPU Affinity", Linux Journal, 2003, url: http://www.linuxjournal.com/article/6799.

[20] The Intel Gigabit Adapters Family, http://support.intel.com/support/network/adapter/1000/

[21] The D-Link DFE-580TX quad network adapter, http://support.dlink .com/products/view.asp?productid=DFE%2D580TX#.

[22] The Agilent N2X Router Tester, http://advanced.comms.agilent.com/n2x/products/.

[23] Oprofile, http://oprofile.sourceforge.net/news/.